姚攀的博客 1.01^365=31.78

Engineers are versatile minds who create links between science, technology, and society.

yyyseb

关注 发私信

访问： 844555次

积分： 8642

等级： BLOG 6

排名： 第2540名

Spring全家桶(八)AOP核心思想与AspectJ 5种类型通知

标签： spring aop Aspectj Aspect

2017-05-17 00:02 1059人阅读 评论(0)

分类：

spring（8）

目录(?) [+]

一、AOP核心思想

AOP是Aspect-Oriented Programming的缩写，翻译为面向切面编程。我个人理解切面就是一个方面。

例子，一个接口里面有增删改查四个方法：

```
1  package com.stuspring.aop.impl;
2
3  /**
4   * Created by bee on 17/5/15.
5   */
6  public interface ArithmeticCalculator {
7       int add(int i,int j);
8       int sub(int i,int j);
9       int mul(int i,int j);
10      int div(int i,int j);
11 }
```

实现类：

```
1  package com.stuspring.aop.impl;
2
3  import org.springframework.stereotype.Component;
4
5  /**
6   * Created by bee on 17/5/15.
7   */
8  @Component("arithmeticCalculator")
```

```
9  public class ArithmeticCalculatorImpl implements ArithmeticCalculator {
10     @Override
11     public int add(int i, int j) {
12         int result=i+j;
13         return result;
14     }
15
16     @Override
17     public int sub(int i, int j) {
18         int result=i-j;
19         return result;
20     }
21
22     @Override
23     public int mul(int i, int j) {
24         int result=i*j;
25         return result;
26     }
27
28     @Override
29     public int div(int i, int j) {
30         int result=i/j;
31         return result;
32     }
33 }
```

如果想给这四个方法分别加上前置日志功能，以其中一个为例,add方法变这样：

```
1   @Override
2     public int add(int i, int j) {
3         System.out.println("The method add begins with " +i+","+j);
4         int result=i+j;
5         return result;
6     }
```

手动给每个方法都加上固然可行，但是维护起来过于麻烦，面向切面编程就是为了解决这一问题。AOP的好处就是每个逻辑位于一个位置，代码不分散便于维护和升级，业务模块更简洁。

加一个日志切面：

```
1  package com.stuspring.aop.impl;
2
3  import org.aspectj.lang.JoinPoint;
4  import org.aspectj.lang.annotation.After;
5  import org.aspectj.lang.annotation.Aspect;
6  import org.aspectj.lang.annotation.Before;
7  import org.springframework.stereotype.Component;
8
9  import java.util.Arrays;
10 import java.util.List;
11
12 /**
13  * Created by bee on 17/5/15.
14  */
```

```
15
16 @Aspect
17 @Component
18 public class LoggingAspect {
19
20     /**
21      * 前置通知 方法开始之前执行
22      * @param joinPoint
23      */
24     @Before("execution(public int com.stuspring.aop.impl.ArithmeticCalculatorIn
25     public void beforeMethod(JoinPoint joinPoint) {
26         String methodName = joinPoint.getSignature().getName();
27         List<Object> agrs = Arrays.asList(joinPoint.getArgs());
28         System.out.println("The method " + methodName + " begins
29     }
30
31     /**
32      * 后置通知，方法执行完之后执行，不论方法是否出现异常
33      * 后置通知中不能访问目标方法的执行结果
34      */
35     @After("execution(public int com.stuspring.aop.impl.ArithmeticCalculatorImp
36     public void afterMethod(JoinPoint joinPoint) {
37         String methodName = joinPoint.getSignature().getName();
38         List<Object> args = Arrays.asList(joinPoint.getArgs());
39         System.out.println("The method " + methodName + " ends with " + args);
40     }
41 }
```

新建spring配置文件beans-aspect.xml：

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xmlns:aop="http://www.springframework.org/schema/aop"
5         xmlns:content="http://www.springframework.org/schema/context"
6         xsi:schemaLocation="http://www.springframework.org/schema/beans
7         http://www.springframework.org/schema/beans/spring-beans.xsd
8         http://www.springframework.org/schema/context
9         http://www.springframework.org/schema/context/spring-context-4.3.xsd
10        http://www.springframework.org/schema/aop
11        http://www.springframework.org/schema/aop/spring-aop-4.3.xsd">
12
13     <content:component-scan base-package="com.stuspring.aop.impl"/>
14     <aop:aspectj-autoproxy/>
15 </beans>
```

附maven依赖：

```
1         <dependency>
2             <groupId>org.springframework</groupId>
3             <artifactId>spring-context</artifactId>
4             <version>4.3.8.RELEASE</version>
5         </dependency>
6
```

```
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>4.3.8.RELEASE</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aop</artifactId>
            <version>4.3.8.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aspects</artifactId>
            <version>4.3.8.RELEASE</version>
        </dependency>

        <dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjrt</artifactId>
            <version>1.8.10</version>
        </dependency>

        <dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjweaver</artifactId>
            <version>1.8.10</version>
        </dependency>
```

二、五种类型通知

1. @Before前置通知：方法开始之前执行。

```
/**
     * 前置通知 方法开始之前执行
     * @param joinPoint
     */
    @Before("execution(public int com.stuspring.aop.impl.ArithmeticCalcul
    public void beforeMethod(JoinPoint joinPoint) {
        String methodName = joinPoint.getSignature().getName();
        List<Object> agrs = Arrays.asList(joinPoint.getArgs());
        System.out.println("The method " + methodName + " begins with " +
    }
```

2. @After后置通知：方法开始之后执行，不论方法是否出现异常。

```
    /**
     * 后置通知，方法执行完之后执行，不论方法是否出现异常
     * 后置通知中不能访问目标方法的执行结果
     */
    @After("execution(public int com.stuspring.aop.impl.ArithmeticCalcula
    public void afterMethod(JoinPoint joinPoint) {
```

```
        String methodName = joinPoint.getSignature().getName();
        List<Object> args = Arrays.asList(joinPoint.getArgs());
        System.out.println("The method " + methodName + " ends with " + a
    }
```

3. @AfterRunning：返回通知，在方法返回结果之后执行。

```
    /**
     * 返回通知，在方法正常结束之后执行的代码
     * 返回通知可以访问方法的返回值
     */

    @AfterReturning(value="execution(public int com.stuspr
    public void afterReturning(JoinPoint joinPoint,Object

        String methodName=joinPoint.getSignature().getName(
        System.out.println("The method "+methodName+" ends with "+result)
    }
```

4. @AfterThrowing：异常通知，在方法抛出异常之后。

```
    /**
     * 异常通知:在方法抛出异常之后执行
     * @param joinPoint
     * @param e
     */

    @AfterThrowing(value="execution(public int com.stuspring.aop.impl.Ari
    public void afterThrowing(JoinPoint joinPoint,Exception e){
        String methodName=joinPoint.getSignature().getName();
        System.out.println("The method "+methodName+" occurs execution: "
    }
```

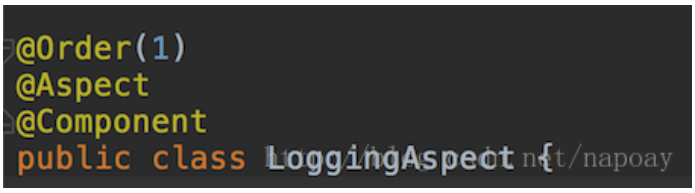
5. @Around：环绕通知，围绕方法执行。

```
    /**
     * 环绕通知需要携带ProceedingJoinPoint类型的参数
     * 环绕通知类似于动态代理的全过程：ProceedingJoinPoint类型的参数可以决定是否执行
     * 环绕通知必须有返回值，返回值即为目标方法的返回值
     *
     * @param pjd
     */
    @Around("execution(public int com.stuspring.aop.impl.ArithmeticCalcul
    public Object aroundMethod(ProceedingJoinPoint pjd) {

        Object result = null;
        String methodName = pjd.getSignature().getName();

        try {
            //前置通知
            System.out.println("--->The method " + methodName + " begins
            //执行目标方法
            result = pjd.proceed();
```

```
19              //后置通知
20              System.out.println("--->The method " + methodName + " ends wi
21
22          } catch (Throwable e) {
23              //异常通知
24              System.out.println("The method "+methodName+" occurs exceptio
25          }
26
27          //后置通知
28          System.out.println("The Method "+methodName+" ends!");
29          return result;
30      }
```

三、指定切面的优先级

切面的优先级可以用@Order注解指定，传入的整数值越小，优先级越高。



四、复用切点表达式

```
1   /**
2    *  定义一个方法用于声明切入点表达式。一般地，该方法不需要再写其它代码。
3    */
4   @Pointcut("execution(public int com.stuspring.aop.impl.ArithmeticCalculato
5   public void declareJoinPointExpression(){
6   }
7   /**
8    * 前置通知 方法开始之前执行
9    * @param joinPoint
10   */
11
12  @Before("declareJoinPointExpression()")
13  public void beforeMethod(JoinPoint joinPoint) {
14      String methodName = joinPoint.getSignature().getName();
15      List<Object> agrs = Arrays.asList(joinPoint.getArgs());
16      System.out.println("The method " + methodName + " begins with " + agrs
17  }
```

外部类引用可以使用报名加方法名的方法。

五、配置文件方式配置AOP

接口：

```
package com.stuspring.aop.fileimpl;

/**
 * Created by bee on 17/5/16.
 */
public interface ArithmeticCalculator {
    int add(int i,int j);
    int sub(int i,int j);
    int mul(int i,int j);
    int div(int i,int j);
}
```

实现类:

```
package com.stuspring.aop.fileimpl;

/**
 * Created by bee on 17/5/16.
 */
public class ArithmeticCalculatorImpl implements ArithmeticCalculator {
    @Override
    public int add(int i, int j) {
        int result=i+j;
        return result;
    }

    @Override
    public int sub(int i, int j) {
        int result=i-j;
        return result;
    }

    @Override
    public int mul(int i, int j) {
        int result=i*j;
        return result;
    }

    @Override
    public int div(int i, int j) {
        int result=i/j;
        return result;
    }
}
```

日志切面:

```
package com.stuspring.aop.fileimpl;

import org.aspectj.lang.JoinPoint;

/**
 * Created by bee on 17/5/16.
 */
```

```
public class LoggingAspect {

    public void beforeMethod(JoinPoint joinPoint){
        String methodName=joinPoint.getSignature().getName();
        System.out.println("The method begins with "+methodName);
    }
}
```

参数验证切面：

```
package com.stuspring.aop.fileimpl;

import org.aspectj.lang.JoinPoint;

/**
 * Created by bee on 17/5/16.
 */
public class ValidationAspect {

    public void validateArgs(JoinPoint joinPoint){
        System.out.println("validationMethod......");
    }
}
```

Spring配置文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/aop
       http://www.springframework.org/schema/aop/spring-aop-4.3.xsd">

    <!--配置bean-->
    <bean id="arithmeticCalculator" class="com.stuspring.aop.fileimpl.Arithmeti
    <bean id="loggingAspect" class="com.stuspring.aop.fileimpl.LoggingAspect"/>
    <bean id="validationAspect" class="com.stuspring.aop.fileimpl.ValidationAsp

    <aop:config>
        <!--配置切面表达式-->
        <aop:pointcut id="pointcut" expression="execution(public int com.stuspr
        <!--配置切面通知-->
        <aop:aspect ref="loggingAspect" order="2">
            <aop:before method="beforeMethod" pointcut-ref="pointcut"/>
        </aop:aspect>

        <aop:aspect ref="validationAspect" order="1">
            <aop:before method="validateArgs" pointcut-ref="pointcut"/>
        </aop:aspect>

    </aop:config>
</beans>
```

Main测试方法：

```
package com.stuspring.aop.fileimpl;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

/**
 * Created by bee on 17/5/16.
 */
public class Main {

    public static void main(String[] args) {
        ApplicationContext ctx=new ClassPathXmlApplicationContex
        ArithmeticCalculator calculator= (ArithmeticCalculator)
        System.out.println(calculator.add(2,4));
    }
}
```


- 上一篇 Spring全家桶(七)通过注解配置Bean
- 下一篇 Spring全家桶(九)Spring JdbcTemplate

查看评论

暂无评论